Scalable Linear Solvers based on Enlarged Krylov subspaces with Dynamic Reduction of Search Directions

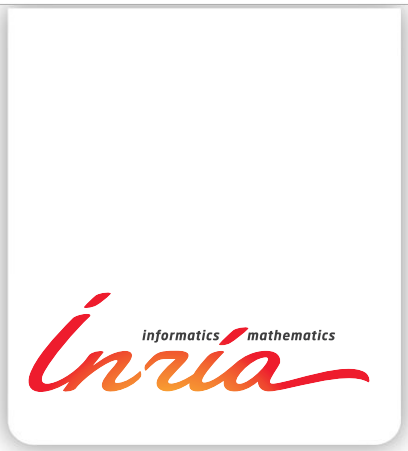
Laura Grigori, Olivier Tissot

Scalable Linear Solvers based on Enlarged Krylov subspaces with Dynamic Reduction of Search Directions

Laura Grigori, Olivier Tissot

RESEARCH REPORT
N° 9190
July 2018
Project-Team Alpines

ISSN 0249-6399 ISRN INRIA/RR--9190--FR+ENG



Scalable Linear Solvers based on Enlarged Krylov subspaces with Dynamic Reduction of Search Directions

Laura Grigori*, Olivier Tissot*

Project-Team Alpines

Research Report n° 9190 — July 2018 — 30 pages

Abstract: Krylov methods are widely used for solving large sparse linear systems of equations. On distributed architectures, their performance is limited by the communication needed at each iteration of the algorithm. In this paper, we study the use of so-called enlarged Krylov subspaces for reducing the number of iterations, and therefore the overall communication, of Krylov methods. In particular, we consider a reformulation of the Conjugate Gradient method using these enlarged Krylov subspaces: the enlarged Conjugate Gradient method.
We present the parallel design of two variants of the enlarged Conjugate Gradient method as well as their corresponding dynamic versions where the number of search directions is dynamically reduced during the iterations. For a linear elasticity problem with heterogeneous coefficients, using a block Jacobi preconditioner, we show that this implementation scales up to 16,384 cores, and is up to 5.7 times faster than the PETSc implementation of PCG.

Key-words: Krylov subspace methods, Conjugate Gradient, Communication reducing algorithms

* INRIA Paris, Sorbonne Université, Université Paris-Diderot SPC, CNRS, Laboratoire Jacques-Louis Lions, ALPINES team (laura.grigori@inria.fr, olivier.tissot@inria.fr.
The work of this author is funded by the NLAFET project as part of European Union's Horizon 2020 research and innovation programme under grant agreement No 671634).

RESEARCH CENTRE
PARIS – ROCQUENCOURT

Domaine de Voluceau, - Rocquencourt
B.P. 105 - 78153 Le Chesnay Cedex

Solveurs linéaires scalables basés sur des sous–espaces de Krylov Élargis avec réduction dynamique des directions de recherche

Résumé : Les méthodes de Krylov sont largement utilisées pour résoudre les systèmes linéaires creux et de grande taille. Sur une architecture distribuée, leur performance est limitée par les communications nécessaires à chaque itération de l'algorithme. Dans ce papier, on étudie l'usage de sous–espaces de Krylov élargis pour réduire le nombre d'itérations, et ainsi le total des communications, des méthodes de Krylov. En particulier, on considère une reformulation de la méthode du Gradient Conjugué qui utilise ces sous–espaces de Krylov élargis : le Gradient Conjugué élargi.

On présente le design parallèle de deux variantes de la méthode du Gradient Conjugué élargi ainsi que les versions dynamiques associées, où le nombre de directions de recherche est réduit dynamiquement pendant les itérations. Pour un problème d'élasticité linéaire avec des coefficients hétérogènes, en utilisant un preconditioneur de type Jacobi par bloc, on montre que cette implémentation passe à l'echelle jusqu'à $16,384$ cœurs, et est jusqu'à $5,7$ fois plus rapide que l'implémentation de PCG présente dans PETSc.

Mots-clés : Méthodes des sous–espace de Krylov, Gradient Conjugué, Algorithmes qui réduisent les communications

1 Introduction

The discretization of partial differential equations, used to model physical phenomena, or optimization problems lead to linear systems of the form $Ax = b$ where A is a sparse matrix. When A becomes very large, iterative methods based on Krylov subspaces are the method of choice [34]. In this paper, we consider the case where $A \in \mathbb{R}^{n\times n}$ is symmetric ($A^\top = A$) positive definite ($x^\top Ax > 0$ for all $x \neq 0$). The Conjugate Gradient method [22], and its preconditioned form, is a well-known method for solving such linear systems.

However, solving these linear systems efficiently on large scale computers remains a challenging problem. One difficulty is the high cost of communication compared to the computation on these machines [10, 9]. Recently, a lot of effort has been put in enhancing the performance of Krylov methods by avoiding global communication [7, 5], overlapping communication with computation [16], or decreasing the number of iterations by searching in multiple directions at once [35, 17]. In this paper, we focus on the third approach, more precisely on the enlarged Conjugate Gradient method (ECG) [17, 19].

After recalling Orthodir and Orthomin variants of ECG, we show the explicit link between the 2 methods. This gives a rigorous justification of an observation already made concerning the robustness of Orthodir compared to Orthomin in [19]. Then we study theoretically the convergence behavior of ECG. We greatly improve the previous result in [17], and show that ECG acts as if the smallest eigenvalues were somehow deflated. Then we present the parallel design of ECG. We consider both Orthodir and Orthomin variants, as well as dynamic versions of these variants that reduce dynamically the number of search directions in order to reduce the extra arithmetic cost in ECG compared to classical CG. In practice, we observe that enlarging the Krylov subspaces can drastically reduce the number of iterations. Indeed in the numerical experiments it is used with a block Jacobi preconditioner and acts as a second-level that, in a way, deflates the smallest eigenvalues; this is in accordance with the theory. This leads to a significant speed-up over the classical PCG. For instance for a 3D linear elasticity problem with heterogeneous coefficients with 4.5 millions of unknowns and 165 millions of nonzero entries, we observe that ECG is up to 5.7 times faster than the PETSc implementation of PCG, both using a block Jacobi preconditioner. This test case is known to be difficult because the classical one-level preconditioners are not expected to be very effective [12]. As it increases the arithmetic intensity and reduces the communication, it is well suited for modern and future architectures that exhibit massive parallelism. For the previous elasticity problem, we show that the method can scale up to $16,384$ threads, each one being bound to one physical core, which means that each core owns nearly 280 unknowns.

In summary, the contributions of the paper are the following. We provide a rigorous justification of the lack of robustness of Orthomin compared to Orthodir observed experimentally in [19]. We give a proof of the speed of convergence of ECG — based on a direct extension of the proof of [6, Theorem 3.2] — which greatly improves the previous existing result presented in [17]. This

shows that ECG acts as a second level preconditioner that mitigates the effect of the smallest eigenvalues on the convergence of the iterative method. Hence it is sufficient to use as preconditioner a highly parallel method as Block Jacobi which bounds the largest eigenvalue of the preconditioned matrix. Finally, we introduce a parallel design embedding several variations of ECG whose scalability is assessed on different matrices and up to $16,384$ cores. We want to point out that our aim is not to design a specific solver for elliptic partial differential equations such as GenEO [37] or multigrid preconditioners with some tuning. For a detailed comparison of such solvers we refer to *Jolivet*'s thesis [28]. It is very likely that for these test cases, these solvers are more effective than ECG with a block Jacobi preconditioner. Nevertheless, unlike these methods ECG is an algebraic method. It does not require any information from the underlying PDE and does not rely on any assumption, except that the matrix is symmetric positive definite. Hence it can be seen as a black-box solver and integrated very easily in any existing code.

2 Enlarged Krylov Conjugate Gradients

2.1 Block Krylov methods

In 1980 *O'Leary* introduced the block Conjugate Gradient method [32] for solving SPD systems with several right-hand sides. In this seminal paper, she proved that block CG can converge significantly faster than CG. This idea was then generalized and extended to other classical Krylov methods as GMRES[33, 31] or BiCGSTAB[15]. Later *Gutknecht*[20] introduced a general framework for defining Block Krylov subspaces.

Recently, Block Krylov methods are receiving an increasing attention in the HPC field [4, 1, 27, 36, 30]. They appear to be well suited for modern computers' architectures with a high level of parallelism because they allow to reduce the number of global synchronizations, while also featuring a higher arithmetic intensity at the cost of some extra computations.

2.2 Enlarged Krylov subspaces

In [17], the authors define so-called enlarged Krylov subspaces. First, the matrix A is reordered by partitioning its graph into $\mathcal{N}$ subdomains (using METIS [29] for example). Then, the initial residual r_0 is split into t vectors denoted $R_0^{e(i)}$, $1 \leq i \leq t$. In the original paper the authors use $t = \mathcal{N}$. It is important to note that the case $t < \mathcal{N}$ can be dealt with many ways as long as $r_0 = \sum_{i=1}^{t} R_0^{e(i)}$ (Fig. 1). This is of particular interest in practice because typically $\mathcal{N}$ will correspond to the number of MPI processes. The parameter t is called the enlarging factor. In practice for a given t the splitting of r_0 does not have a high impact on the convergence of the method. In the numerical experiments we construct the initial enlarged residual $R_0^e = [R_0^{e(1)}, \ldots, R_0^{e(t)}]$ as the leftmost example in Fig. 1.

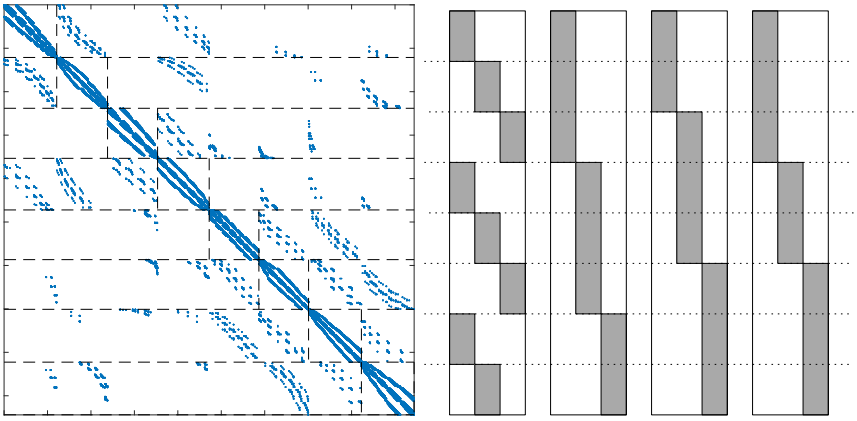


Figure 1: Illustration of the ordering of A into 8 subdomains obtained with METIS [29] and several admissible splittings of r_0 into 3 vectors.

Then, the enlarged Krylov subspace of order k denoted $\mathcal{K}_{k,t}(A, r_0)$ is defined as the block Krylov subspace of order k associated to A and the enlarged residual R_0^e. More precisely, and following the notation introduced in [20],

$$\mathcal{K}_{k,t}(A, r_0) := \mathcal{K}_k^{\square}(A, R_0^e) \tag{1}$$

$$= \text{span}^{\square}\left\{R_0^e, AR_0^e, \ldots, A^{k-1}R_0^e\right\}. \tag{2}$$

Using this definition and following [19] it is possible to derive two variants (Orthomin and Orthodir) of the enlarged Conjugate Gradient (ECG) algorithm (Fig. 1). More precisely, the enlarged approximate solution is a matrix of size $n \times t$ denoted X_k, and the sum of its columns gives the approximate solution of the original system. We denote R_k the enlarged approximate residual, and similarly we obtain the approximate residual of the original system by summing its columns. P_k is a matrix of size $n \times t$ called search directions, it corresponds to the A-orthonormalization of Z_k. We denote α_k the optimal step, unlike in CG algorithm it is not a scalar but a matrix of size $t \times t$. Depending on the method for constructing Z_{k+1}, it is possible to derive two variants of ECG: Orthomin and Orthodir.

Orthomin (Omin) corresponds to Block CG [32]:

$$\beta_k = (AP_k)^{\top} R_k, \tag{3}$$

$$Z_{k+1} = R_k - P_k\beta_k. \tag{4}$$

This method is very similar to the one originally proposed by *Hestenes and Stiefel* [22] because it constructs the new descent directions Z_{k+1} using R_k and P_k.

Orthodir (Odir) corresponds to the Block Lanczos algorithm but with the

inner product induced by A:

$$\gamma_k = (AP_k)^\top (AP_k), \tag{5}$$

$$\rho_k = (AP_{k-1})^\top (AP_k), \tag{6}$$

$$Z_{k+1} = AP_k - P_k\gamma_k - P_{k-1}\rho_k. \tag{7}$$

It is the block equivalent of the homonym method defined in [2]. Unlike the previous variant, Z_{k+1} is constructed using P_k and P_{k-1}.

Both Orthodir and Orthomin produce Z_{k+1} that is A-orthogonal to P_i for $i \leq k$. Then the search directions P_{k+1} are defined as

$$P_{k+1} = Z_{k+1}(Z_{k+1}^\top AZ_{k+1})^{-1/2}. \tag{8}$$

Unlike CG algorithm a breakdown would occur if $Z_{k+1}^\top AZ_{k+1}$ is singular, *i.e.*, Z_{k+1} is not full rank. Although rare this situation can happen in practice and several variants have been developed in order to handle this case [25, 19, 14, 32]. Overall, both Orthomin and Orthodir generate P_{k+1} such that

$$P_{k+1}^\top AP_i = 0, \forall i \leq k \tag{9}$$

$$P_{k+1}^\top AP_{k+1} = I. \tag{10}$$

Consequently, the ECG method can be summarized in Algorithm 1. Another difference with the original block CG algorithm is that the search directions are A-orthonormalized at each iteration: P_k is used as search directions instead of Z_k. It has be shown numerically that using this variant can increase the numerical stability of the method [14].

Given a preconditioner M^{-1}, the idea for applying left preconditioning to the (block) Conjugate Gradient method is to remark that $M^{-1}A$ is self-adjoint with respect to the M-inner product [34]. Then by replacing A by $M^{-1}A$, and the transpose by ${}^\top M$ in the algorithm (Fig. 1), it follows the preconditioned enlarged Conjugate Gradient method. In fact, some simplifications occur and the algorithm remains exactly the same except the definition of Z_k that is slightly different. More precisely, it follows that the preconditioned Orthomin method corresponds to,

$$Z_{k+1} = (I - P_kP_k^\top A)M^{-1}R_k, \tag{11}$$

and the preconditioned Orthodir method corresponds to,

$$Z_{k+1} = (I - P_kP_k^\top A - P_{k-1}P_{k-1}^\top A)M^{-1}AP_k. \tag{12}$$

In both cases, the initialization also slightly differs because $Z_1 = M^{-1}R_0^e$. Overall, the preconditioner is applied once per iteration, as in the classical CG method.

2.3 Equivalence between Orthodir and Orthomin

In what follows, we assume exact arithmetic and we study the connection between these two methods with the aim to derive formulas that links the approximate quantities of both variants. Indeed, by construction the approximate

Algorithm 1 ECG algorithm.

$P_0 = 0$
$Z_1 = R_0^e$
$k = 1$
for $k = 1, \ldots, k_{\max}$ **do**
  $P_k = Z_k(Z_k^\top A Z_k)^{-1/2}$
  $\alpha_k = P_k^\top R_{k-1}$
  $X_k = X_{k-1} + P_k \alpha_k$
  $R_k = R_{k-1} - A P_k \alpha_k$
  if $||\sum_{i=1}^t R_k^{(i)}||_2 < \varepsilon$ **then**
    stop
  end if
  construct Z_{k+1} using (3)-(4) (Orthomin) or (5)-(7) (Orthodir)
  $k = k + 1$
end for
$x_k = \sum_{i=1}^t X_k^{(i)}$

solutions computed by Orthodir and Orthomin are equal. Hence, the approximate residuals are also equal. But this does not imply that the search directions generated are equal even if they belong to the same space. We denote with a tilde the variables related to Orthomin and with a hat the variables related to Orthodir, *e.g.*, $\widehat{P}_k$ are the A-orthonormalized search directions generated during Orthodir. In order to simplify the presentation we consider that no breakdowns have occurred, *i.e.*, $\widehat{Z}_k$, $\widehat{P}_k$ and $\widetilde{Z}_k$, $\widetilde{P}_k$ are all well-defined.

Since Orthomin and Orthodir rely on the same projection process [19] (they both search an approximate solution in $\mathcal{K}_{t,k}$, such that the corresponding residual is orthogonal to $\mathcal{K}_{t,k}$), we know that $\widehat{X}_k = \widetilde{X}_k$. It follows that:

$$\widehat{P}_k \widehat{\alpha}_k = \widetilde{P}_k \widetilde{\alpha}_k, \tag{13}$$

$$\widehat{P}_k \widehat{P}_k^\top = \widetilde{P}_k \widetilde{P}_k^\top. \tag{14}$$

Hence, there exists $\delta_k \in \mathbb{R}^{t \times t}$ orthogonal and such that $\widetilde{P}_k = \widehat{P}_k \delta_k$. By definition we also have,

$$R_k - R_{k-1} = -A\widetilde{P}_k \widetilde{\alpha}_k. \tag{15}$$

A simple computation using the previous relationships gives,

$$-\widehat{Z}_{k+1}\delta_k \widetilde{\alpha}_k = -A\widetilde{P}_k \widetilde{\alpha}_k + \widetilde{P}_k \widetilde{\gamma}_k \widetilde{\alpha}_k + \widetilde{P}_{k-1}\widetilde{\rho}_k \widetilde{\alpha}_k, \tag{16}$$

$$= R_k - \widetilde{P}_k \widetilde{\beta}_k - \widetilde{Z}_k + \widetilde{P}_k \widetilde{P}_k^\top A R_{k-1}, \tag{17}$$

and,

$$\widetilde{P}_k \widetilde{P}_k^\top A R_{k-1} = \widetilde{Z}_k (\widetilde{Z}_k^\top A \widetilde{Z}_k)^{-1} \widetilde{Z}_k^\top A R_{k-1}, \tag{18}$$

$$= \widetilde{Z}_k (\widetilde{Z}_k^\top A \widetilde{Z}_k)^{-1} \widetilde{Z}_k^\top A \widetilde{Z}_k, \tag{19}$$

$$= \widetilde{Z}_k. \tag{20}$$

Thus,

$$\widetilde{Z}_{k+1} = -\widehat{Z}_{k+1}\delta_k\widetilde{\alpha}_k. \tag{21}$$

This result is a generalization of a previous result presented by *Manteuffel et al.* [2, p. 1550] for the classical CG. In fact, the authors show that $\widetilde{z}_k = \Pi_{i=0}^{k}(-\widetilde{\alpha}_k)\widehat{z}_k$ but they never consider explicitly the A-orthonormalized search directions. In particular, they define z_{k+1} using z_k (for Omin) and z_{k-1} (for Odir). This explains the slight difference between our generalization and their result.

When k becomes large, $\widetilde{\alpha}_k = \widetilde{P}_k^\top R_{k-1}$ and $||\widetilde{\alpha}_k||_2$ is more likely to be low because R_{k-1} is supposed to converge to 0 and $\widetilde{P}_k$ is A-orthonormalized – the same reasoning applies for $\widehat{\alpha}_k$. This result is very interesting because it shows that, since δ_k is an orthogonal matrix, when k becomes large $||\widetilde{Z}_{k+1}||_2$ can be significantly lower than $||\widehat{Z}_{k+1}||_2$. Hence, the conditioning of $\widetilde{Z}_{k+1}^\top A\widetilde{Z}_{k+1}$ could be much worse than that of $\widehat{Z}_{k+1}^\top A\widehat{Z}_{k+1}$, possibly leading to a breakdown when computing its Cholesky factorization (line 5 in Fig. 1). It is remarkable to notice that even for classical CG method, this has already been noticed by *Manteuffel et al.* in [2, p. 1551-1552]: "If BCA is indefinite, Omin may still be used, but the previous direction vector [...] should be stored. Then, if $\widehat{\alpha}_i = 0$, control can switch to the 3-term recursion of Odir to get p_{i+1}". In practice, this phenomenon is indeed observed: there are cases where Orthomin breaks down while Orthodir does not [19]. In conclusion, Orthodir is expected to be more reliable than Orthomin. However, Orthodir is also more costly than Orthomin: the construction of Z_{k+1} requires twice as many flops and memory as for Orthomin.

2.4 Convergence study

As previously mentioned, *O'Leary* [32] proved that block CG can converge significantly faster than the classical CG. In [17], it is proved that ECG converges faster than CG but there is no further information on the speed of convergence of ECG. This section is dedicated to the proof of the following result.

Theorem 1 *Let x_k be the approximate solution given by the Enlarged Conjugate Gradient method with an enlarging factor t at step k. Then we have:*

$$||x_k - x_*||_A^2 \leq C\left(\frac{\sqrt{\kappa_t}-1}{\sqrt{\kappa_t}+1}\right)^{2k} \tag{22}$$

where $\kappa_t = \dfrac{\lambda_n}{\lambda_t}$ and C is a constant independent of k whose exact expression will be given in the course of the proof.

Before starting the proof, we want to point out that this result is a big improvement of the theorem stated in [17] because it explains that ECG's convergence is, to some extent, closer to that of Deflated-CG[13] rather than that of the classical CG.

Proof 1 *The key idea of the proof is to remark the close link between ECG and block CG. In particular, we follow the proof of Theorem 3.2 by* Jie Chen *in [6] for block CG, and we adapt it to our case of interest: ECG. The approach of the proof is similar to that of* O'Leary *[32] but the final result is slightly different.*

First, we need to write the error at iteration k as a polynomial of A evaluated in the initial error. Indeed,

$$x_k = x_0 + \sum_{j=1}^{t} q_{kj}(A) A d_0^{(j)}, \tag{23}$$

where $d_0^{(j)}$ is the i-th component of the enlarged initial error [6], and q_{kj} is a polynomial of degree not exceeding $k-1$.

Hence,

$$e_k = x_k - x^* = \sum_{j=1}^{t} (1 + q_{kj}(A)A) d_0^{(j)} \tag{24}$$

$$= \sum_{j=1}^{t} p_{kj}(A) d_0^{(j)}, \tag{25}$$

where $p_{kj}(X) = 1 + q_{kj}(X)X$ is a polynomial of degree not exceeding k and such that $p_{kj}(0) = 1$, $\forall k, j$.

Let $A = \Phi^\top \Lambda \Phi$ be the spectral decomposition of A. Let us rewrite e_k according to this decomposition,

$$e_k = \sum_{j=1}^{t} p_{kj}(A) d_0^{(j)} = \sum_{j=1}^{t} \Phi p_{kj}(\Lambda) \Phi^\top d_0^{(j)}. \tag{26}$$

Thus,

$$||e_k||_A^2 = e_k^\top A e_k \tag{27}$$

$$= \left(\sum_{j=1}^{t} \Phi p_{kj}(\Lambda) \Phi^\top d_0^{(j)} \right)^\top A \left(\sum_{j=1}^{t} \Phi p_{kj}(\Lambda) \Phi^\top d_0^{(j)} \right) \tag{28}$$

$$= \left(\sum_{j=1}^{t} {d_0^{(j)}}^\top \Phi p_{kj}(\Lambda) \right) A \left(\sum_{j=1}^{t} p_{kj}(\Lambda) \Phi^\top d_0^{(j)} \right). \tag{29}$$

This final expression is a generalization of the expression that occurs in the proof of convergence of the CG algorithm in [34].

Let $\widehat{j} \in \{1, \ldots, t\}$. We denote $p := p_{k\widehat{j}}$ in order to simplify the notations. Following Chen *[6], let us define*

$$-p(\Lambda)^{-1}(2I - p(\Lambda))\Phi^\top d_0^{(j)} = \begin{pmatrix} F_1 \\ F_2 \end{pmatrix}, \forall j \neq \widehat{j}, \tag{30}$$

$$\Phi^\top d_0^{(\widehat{j})} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}. \tag{31}$$

The other polynomials p_{kj} are chosen such that $p_{kj} = \tau_j(2-p)$ where the τ_j are defined as the components of the solution of $F_1\tau = f_1$.

From those definitions, it follows that for $j \neq \widehat{j}$

$$p_{kj}(\Lambda)\Phi^\top d_0^{(j)} = (2I - p(\Lambda))\Phi^\top d_0^{(j)}\tau_j \tag{32}$$

$$= -p(\Lambda)(-p(\Lambda)^{-1}(2I - p(\Lambda))\Phi^\top d_0^{(j)})\tau_j \tag{33}$$

$$= -p(\Lambda)\begin{pmatrix} F_1 \\ F_2 \end{pmatrix}\tau_j. \tag{34}$$

Hence, when summing up those terms

$$\sum_{j\neq\widehat{j}} p_{kj}(\Lambda)\Phi^\top d_0^{(j)} = -p(\Lambda). \tag{35}$$

Thus,

$$\sum_{j=1}^{t} p_{kj}(\Lambda)\Phi^\top d_0^{(j)} = p(\Lambda)\begin{pmatrix} f_1 \\ f_2 \end{pmatrix} - p(\Lambda)\begin{pmatrix} f_1 \\ F_2F_1^{-1}f_1 \end{pmatrix} \tag{36}$$

$$= \begin{pmatrix} 0 & 0 \\ -p(\Lambda_2)F_2F_1^{-1} & p(\Lambda_2) \end{pmatrix}\Phi^\top d_0^{(\widehat{j})} \tag{37}$$

$$:= \begin{pmatrix} 0 & 0 \\ E & P \end{pmatrix}\Phi^\top d_0^{(\widehat{j})}. \tag{38}$$

Finally,

$$||e_k||_A^2 = {d_0^{(\widehat{j})}}^\top \Phi \begin{pmatrix} 0 & E^\top \\ 0 & P \end{pmatrix} A \begin{pmatrix} 0 & 0 \\ E & P \end{pmatrix}\Phi^\top d_0^{(\widehat{j})} \tag{39}$$

$$\leq ||d_0^{(\widehat{j})}||_A^2 \left|\left|\begin{pmatrix} E^\top E & E^\top P \\ PE & P^2 \end{pmatrix}\right|\right|. \tag{40}$$

Let us focus on the right term in this last inequality,

$$\left|\left|\begin{pmatrix} E^\top E & E^\top P \\ PE & P^2 \end{pmatrix}\right|\right| \leq ||P||^2 \left|\left|\begin{pmatrix} 0 & 0 \\ F_2F_1^{-1} & I \end{pmatrix}\right|\right|^2 \tag{41}$$

$$\leq ||P||^2\left(a^2+1\right) \tag{42}$$

where a is the largest singular value of $F_2F_1^{-1}$.

Finally, we have

$$||e_k||_A^2 \leq ||d_0^{(\widehat{j})}||_A^2\left(a^2+1\right)||P||^2. \tag{43}$$

If we replace p (and therefore P) by the optimal choice we can rewrite the bound as the following min-max problem,

$$||e_k||_A \leq ||d_0^{(\widehat{j})}||_A\sqrt{(a^2+1)}\min_{p\in\mathbb{P}_1^k}\max_{t\leq i\leq n}|p(\lambda_i)|. \tag{44}$$

Finally, it is possible to use Chebyshev polynomials to estimate this min-max quantity [32, 34],

$$\min_{p\in\mathbb{P}_1^k} \max_{t\leq i\leq n} |p(\lambda_i)| \leq 2\left(\frac{\sqrt{\kappa_t}-1}{\sqrt{\kappa_t}+1}\right)^k. \tag{45}$$

2.5 Dynamic reduction of the search directions

In what follows, we recall an approach for reducing the block size in the Orthodir method during the iterations presented in [19]. The idea is to reduce the arithmetic and memory extra costs of Orthodir while maintaining its good convergence behavior. As explained in the survey [20] the key idea to reduce the block size is to monitor the rank of R_k. Once R_k becomes rank deficient, it means that a part of the approximate solution has already converged at iteration k. More precisely for $i \geq k-1$, there exists a linear combination (independent of i) of columns of X_i that remains constant. As a consequence, there exists a linear combination of search directions that is not useful anymore to compute the approximate solution. The idea is to remove these search directions in the next iterations. As R_{k-1} is an $n \times t$ matrix with n large, it is preferable to avoid computing the rank of R_{k-1} directly. In [19], it is shown that the rank of $\alpha_k = P_k^\top R_{k-1}$ can be computed instead.

The method presented in [19] can be divided into two parts. At each iteration of the algorithm (Fig. 2) a Singular Value Decomposition of α_k is computed (line 8). If the numerical rank of α_k is below a given tolerance then the search directions are reduced accordingly (line 12) and some of them are kept in order to keep the A-orthogonality property (line 14, 24 and 25). Although computing the SVD of α_k at each iterations induces an extra cost compared to Orthodir, this operation does not involve any communications and it is negligible because α_k is a small matrix of size $t \times t$. Furthermore, as the search directions P_k are reduced, the dominant operation of Krylov iterations in terms of flops; the matrix product (AP_k), and the application of the preconditioner ($M^{-1}AP_k$), are cheaper.

2.6 Curing breakdowns in Orthomin

As explained previously Orthomin version of ECG can break down. There exist several methods to overcome this issue and in the following we recall the *Breakdown-Free block CG* method defined in [25]. Starting from the original algorithm of *O'Leary* [32], the authors propose to perform a rank-revealing QR decomposition of Z_{k+1} and then drop its null part before A-orthonormalizing it. They show that in exact arithmetic this allows to continue the algorithm with nearly no further modification. The resulting algorithm is given in Algorithm 3.

From a practical point of view the size of P_{k+1} can be reduced, but at each iteration Z_{k+1} is of size $n \times t$ because the size of R_k remains constant. Hence the matrix product AP_k is cheaper but the application of the preconditioner

Algorithm 2 ECG D-Odir algorithm.

$P_0 = 0$
$Z_1 = R_0^e$
$H = \emptyset$
$k = 1$
for $k = 1, \ldots, k_{\max}$ **do**
  $P_k = Z_k(Z_k^\top A Z_k)^{-1/2}$
  $\alpha_k = P_k^\top R_{k-1}$
  $\alpha_k = U_k \Sigma_k V_k^\top$
  let s_k be the number of singular values of α_k bigger than ε_{def}
  if $s_k < s_{k-1}$ **then**
    $\alpha_k = U_k^\top \alpha_k$
    $P_k = P_k U_k$
    $\alpha_k = \alpha_k(1 : s_k, :)$
    $H = [H, P(:, s_k : s_{k-1})]$
    $P_k = P_k(:, 1 : s_k)$
  end if
  $X_k = X_{k-1} + P_k \alpha_k$
  $R_k = R_{k-1} - A P_k \alpha_k$
  if $||\sum_{i=1}^{t} R_k^{(i)}||_2 < \varepsilon$ **then**
    stop
  end if
  $\gamma_k = (AP_k)^\top (AP_k)$
  $\rho_k = (AP_{k-1})^\top (AP_k)$
  $\delta_k = (AH)^\top (AP_k)$
  $Z_{k+1} = AP_k - P_k \gamma_k - P_{k-1} \rho_k - H \delta_k$
  $k = k + 1$
end for
$x_k = \sum_{i=1}^{t} X_k^{(i)}$

$M^{-1}R_k$ is not. Furthermore computing a rank-revealing QR factorization of Z_{k+1} cannot be neglected because Z_{k+1} is of size $n \times t$. In summary, as this method was not meant for efficiency, but rather for improving the stability of Orthomin, it does not allow to save as many computations as in the dynamic variant of Orthodir.

Algorithm 3 BF-ECG algorithm.

$P_0 = 0$
$Z_1 = R_0^e$
$k = 1$
for $k = 1, \ldots, k_{\max}$ **do**
  $P_k = Z_k(Z_k^\top A Z_k)^{-1/2}$
  $\alpha_k = P_k^\top R_{k-1}$
  $X_k = X_{k-1} + P_k \alpha_k$
  $R_k = R_{k-1} - A P_k \alpha_k$
  if $||\sum_{i=1}^{t} R_k^{(i)}||_2 < \varepsilon$ **then**
    stop
  end if
  $\beta_k = (AP_k)^\top R_k$
  $Z_{k+1} = R_k - P_k \beta_k$
  $Z_{k+1} = ortho(Z_{k+1})$
  $k = k + 1$
end for
$x_k = \sum_{i=1}^{t} X_k^{(i)}$

3 Parallel design

3.1 Data distribution

As it is usually the case in parallel implementations of Krylov methods, we assume that the unknowns are distributed among the processors. We also assume that each processor owns different unknowns. Thus all the variables whose size scales as the size of the linear system (X_k, R_k, P_k, AP_k, Z_k) are distributed row wise among the processors according to the distribution of the unknowns. All variables whose size scales as the enlarging factor t (α_k, β_k, γ_k, ρ_k) are replicated on all the processors. Locally they are stored contiguously and column by column. There is no allocation or deallocation of memory during the iterations. In particular, when using Dynamic Orthodir or Breakdown-Free Orthomin the memory is not freed when the block size is reduced. The local memory consumption of preconditioned Orthodir and Orthomin on P processors is summarized in Table 1. For completeness, we also add the local memory consumption of the classical CG algorithm, described in [34] for instance, where only 5 vectors and 2 scalars are needed.

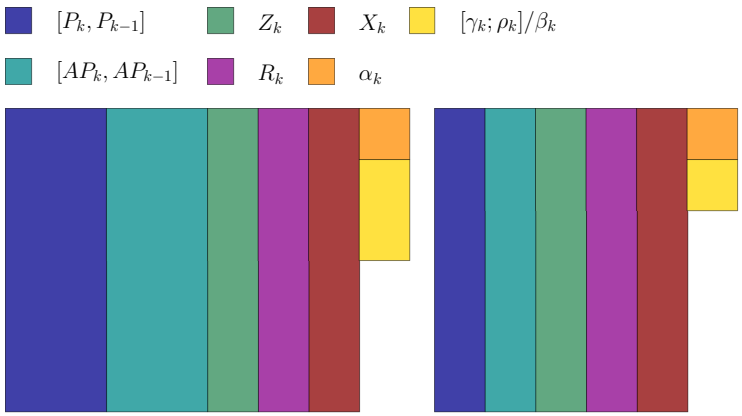


Figure 2: Local distribution of the data: Orthodir on the left and Orthomin on the right.

3.2 Cost analysis of ECG

Our implementation of ECG is based on Reverse Communication Interface [24]. For one iteration of ECG, it requires external routines to apply the sparse matrix product and the preconditioner to a set of vectors. Indeed the implementation of these routines highly depends on the linear system to be solved. This is why we do not take into account these operations in our cost analysis.

Given n, t such that $t \ll n$, we denote V, W tall and skinny matrices of size $n \times t$ whose rows are distributed among the processors, and α is a matrix of size $t \times t$ replicated on the P processors. Following [30], it is possible to decompose the iterations of ECG (and more generally block CG) into the following kernels:

- $V \leftarrow V + W\alpha$ (`tsmm` in [30]),
- $\alpha \leftarrow V^\top W$ (`tsmtsm` in [30]),
- Cholesky factorization of α (`potrf`),
- triangular solve of α with several right-hand sides (`trsm`).

Following ECG algorithm (fig. 1), each iteration of Orthodir and Orthomin consists of 3 `tsmm` (lines 7, 8, and 12), 4 `tsmtsm` (lines 5, 6, 9, and 12), 1 `potrf` (line 5) and 2 `trsm` (line 5). Indeed, the line 5 of the algorithm (fig. 1) can be decomposed as,

$AZ_k \leftarrow A * Z_k$	*sparse matrix set of vectors*
$C \leftarrow$ `tsmtsm`(Z_k, AZ_k)	*form* $Z_k^\top AZ_k$
$C \leftarrow$ `potrf`(C)	*Cholesky factorization*
$P_k \leftarrow$ `trsm`(Z_k, C)	
$AP_k \leftarrow$ `trsm`(AZ_k, C)	*update* P_k *and* AP_k

	# flops	# messages	# words	memory
Omin	$16\frac{nt^2}{P} + 4\frac{nt}{P} + \frac{1}{3}t^3$	$4\log_2(P)$ (4)	$4t^2$	$5\frac{nt}{P} + 2t^2$
Odir	$20\frac{nt^2}{P} + 5\frac{nt}{P} + \frac{1}{3}t^3$	$4\log_2(P)$ (4)	$5t^2$	$7\frac{nt}{P} + 3t^2$
CG	$10\frac{n}{P}$	$2\log_2(P)$ (2)	2	$5\frac{n}{P}$

Table 1: Complexity and memory consumption of Orthodir, Orthomin and CG where t is the enlarging factor, n is the number of rows of A and P is the number of processors. In parenthesis is indicated the number of calls to `MPI_Allreduce`.

Doing so allows us to avoid calling the sparse matrix set of vectors product for computing AP_k at the price of an extra `trsm`. Hence the difference between the two algorithms is the construction of Z_{k+1} (line 12). The `tsmtsm` and `tsmm` for constructing Z_{k+1} in Orthodir (equations (5)-(7)) are twice as much costly as for Orthomin (equations (3)-(4)).

As matrices of size $t \times t$ are replicated among the processors, we notice that `tsmm`, Cholesky factorization of α and triangular solve of α are local operations without any communication. Hence we use the corresponding LAPACK routines: `gemm`, `potrf` (dense Cholesky factorization) and `trsm` (dense triangular solve with several right-hand sides). However V and W are distributed and `tsmtsm` is not a local operation. The LAPACK routine `gemm` is called to compute the local product $V_i^\top W_i$ followed by a call to `MPI_Allreduce`.

Thus, the only kernel operation that requires a communication is `tsmtsm` and 4 calls to `MPI_Allreduce` are done per iteration. It is usually assumed that during a call to `MPI_Allreduce` the number of messages sent and received on the network is equal to $\log_2(P)$ – although the exact number depends on the MPI implementation [38]. Moreover it is a blocking operation: when completed all the processors are synchronized. This is why in practice, as in the classical CG, the communication cost is dominated by 2 calls to `MPI_Allreduce`: the one after the sparse matrix set of vectors product (line 5) and the one after the preconditioner (line 12), because they occur after operations with a potential load imbalance between processors.

In summary, the detailed costs of one iteration of Orthodir and Orthomin in terms of flops, words, and messages are indicated in Table 1. For the sake of comparison, we recall the complexity of the CG algorithm described in [34]. We also report the number of `MPI_Allreduce` in parenthesis, in addition to the order of magnitude of the number of messages. In summary, one iteration of ECG is approximately t^2 times more costly in terms of flops than one iteration of CG. While the number of messages is of the same order, the number of words is also t^2 times larger. Indeed there is a trade-off between these extra costs and the reduction of the number of iterations due to the enlargement of the search spaces.

3.3 Cost of dynamic reduction of ECG

The implementation of the dynamic reduction of the search directions within Orthodir follows Algorithm 2. In practice, we use LAPACK routine `gesvd` and only compute the right singular vectors of α_k denoted U_k. We check the singular values obtained. If there are some lower than $\frac{\varepsilon}{\sqrt{t}}$, which is the criterion proposed in [19], we call `geqrf` on U in order to perform the updates PU_k, APU_k and $U_k^\top \alpha_k$ in-place with `ormqr`. Since P_k and AP_k are stored in a column major fashion the selection of the columns is done at no cost. Similarly H is not explicitly defined. However the selection of the first rows of α_k implies an in-place memory rearrangement.

The implementation of Breakdown-Free Orthomin is similar to Orthomin except the computation of a rank-revealing QR decomposition of Z_{k+1}. As Z_{k+1} is distributed, it is not reasonable to use a LAPACK kernel to compute it. Instead we use a modification of Chol-QR algorithm [40] which is a cheaper but less stable alternative to TS-RRQR [11, 9]. Its implementation is very easy using the LAPACK routine `pstrf` (Cholesky with pivoting) for computing (R, π) at line 2. Following [25] we use the default tolerance of `pstrf` for detecting exact rank deficiency of Z_{k+1}.

Algorithm 4 Chol-RRQR

Input: P, ε
Output: Q_1 orthogonal such that

$$P\pi = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$$

where π is a permutation and all the diagonal elements of R_{11} are larger than ε

1: $\mu \leftarrow P^\top P$
2: Compute (R, π) such that $\pi^\top \mu \pi = R^\top R$ with $R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$ and all the diagonal elements of R_{11} are larger than ε^2
3: $P_1 \leftarrow P\pi(:, 1 : size(R_{11}))$
4: $Q_1 \leftarrow P_1 R_{11}^{-1}$

4 Numerical experiments

4.1 Description of the parallel environment

In the experiments we use a block Jacobi preconditioner, associating at each block a MPI process. Before calling ECG, each MPI process factorizes the diagonal block of A corresponding to the local row panel that it owns. At each iteration of ECG, each MPI process performs a backward and forward solve

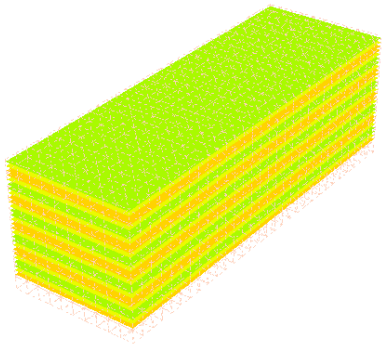


Figure 3: Heterogeneity pattern of the Young's modulus and Poisson's ratio for elasticity matrices.

locally in order to apply the preconditioner. Hence the application of the preconditioner does not need any communication. It is likely that there exists better preconditioners than block Jacobi for our test cases, however we are interested in the iterative method rather than in the preconditioner. In particular, we do not want to target specific applications and aim at being as generic as possible. Although in theory it is possible to apply any preconditioner within this implementation, in practice it is essential that applying this preconditioner to several vectors at the same time is not too costly, *e.g*, a sublinear complexity with respect to the number of vectors.

The following experiments are performed on a machine located at Umeå University as part of High Performance Computing Center North (HPC2N), called Kebnekaise. It is a heterogeneous machine formed by a mix of Intel Xeon E5-2690v4 (Broadwell) with 2x14 cores (and E7-8860v4 for large memory computations), Nvidia K80 GPU and Intel Xeon Phi 7250 (Knight's Landing) with 68 cores. In our experiments, we use the so-called compute nodes, which are formed by Intel Xeon E5-2690v4 (Broadwell) with 2x14 cores. For a detailed description of the machine, we refer to the online documentation[1].

We compile the code (and its dependencies) using Intel toolchain installed on the machine: `mpiicc` (based on `icc` version 18.0.1 20171018) and MKL [39] version 2018.1.163. We use PETSc [3] in order to compare ECG implementation to PETSc PCG implementation. In particular, PETSc is configured to use MKL-PARDISO as exact solver for sparse matrices in the block Jacobi preconditioner. For partitioning the matrix we are using the METIS library downloaded and installed by PETSc.

[1] `https://www.hpc2n.umu.se/resources/hardware/kebnekaise`

Name	Size	Nonzeros	Problem
Flan_1565	1,564,794	117,406,044	Structural problem
Bump_2911	2,911,419	130,378,257	Reservoir simulation
Ela_20	2,118,123	74,735,397	Linear elasticity
Ela_30	4,615,683	165,388,197	Linear elasticity

Table 2: Test matrices.

4.2 Test cases

The Ela matrices arise from the linear elasticity problem with Dirichlet and Neumann boundary conditions defined as follows

$$\operatorname{div}(\sigma(u)) + f = 0 \qquad \text{on } \Omega \tag{46}$$

$$u = 0 \qquad \text{on } \partial\Omega_D \tag{47}$$

$$\sigma(u) \cdot n = 0 \qquad \text{on } \partial\Omega_N \tag{48}$$

where Ω is a unit cube. The matrices Ela_N correspond to this equation discretized with FreeFem++ [21] using a triangular mesh with $1600 \times N \times N$ points on the corresponding vertices and P1 finite elements scheme. $\partial\Omega_D$ is the Dirichlet boundary, $\partial\Omega_N$ is the Neumann boundary, f is some body force, u is the unknown displacement field. $\sigma(.)$ is the Cauchy stress tensor given by Hooke's law: it can be expressed in terms of Young's Modulus E and Poisson's ratio ν. For a more detailed description of the problem see [26, 18, 36]. We consider a heterogeneous beam made of several layers of a hard material $(E_1, \nu_1) = (2 \times 10^{11}, 0.25)$ and a soft material $(E_2, \nu_2) = (10^7, 0.45)$, *i.e.*, discontinuous E and ν (Figure 3). This test case is known to be difficult because the matrix is ill conditioned. In particular, the classical one-level preconditioners are not expected to be very effective [12].

As previously pointed out, ECG is an algebraic method that does not rely on any particular assumption on the matrix, except that it is symmetric positive definite. As an illustration, we also test the implementation on two SPD matrices coming from the Sparse Matrix Collection of *Tim Davis* [8]: Flan_1564 and Bump_2911. Numerical properties of the test matrices are summarized in Table 2.

4.3 Results

In all the experiments the tolerance is set as the default tolerance of PETSc, *i.e.*, 10^{-5} and the maximum number of iterations is set to 5000. The right-hand side is chosen uniformly random and normalized and the initial guess is set to 0. We do not use any kind of threading and use 28 MPI processes per node. Unless otherwise stated, we use one OpenMP thread per MPI process — we

also perform numerical experiments to observe the effect of threading in the last section.

4.3.1 Impact of the enlarging factor

First we study the impact of the enlarging factor t on the methods. We fix the number of processors and we vary the value of t for the 4 methods: Orthodir (Odir), Orthodir with dynamic reduction of the search directions (D-Odir), Orthomin (Omin) and Breakdown-Free Orthomin (BF-Omin). The results obtained are summarized in Table 3

For Flan_1565 the number of MPI processes is fixed to 56. We remark that the runtime is decreasing until $t = 12$ and then it increases slightly. When t is relatively small the 4 methods are comparable. However as t increases the effect of dynamic reduction becomes more visible. With $t = 28$, D-Odir is almost 10% faster than Odir. On the other hand, as we are detecting exact rank deficiency of Z_{k+1}, BF-Omin did not reduce the size of the block and as no breakdown occurs it is slightly more costly than Omin. We also tested $\frac{\varepsilon}{\sqrt{t}}$ as the tolerance for detecting breakdowns but this does not allow the method to converge. Overall, for this matrix, the best method is D-Odir with $t = 12$.

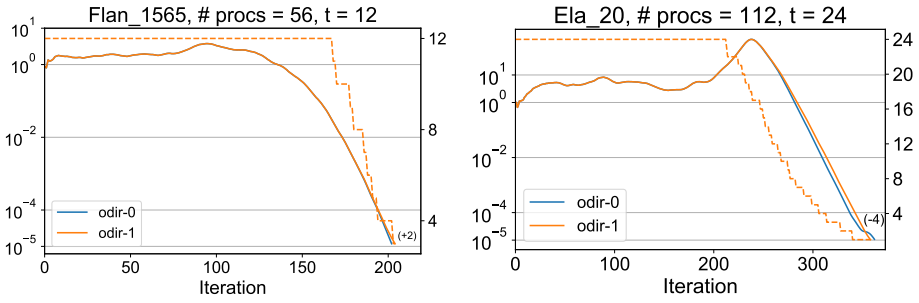
For Bump_2911 we fix the number of MPI processes to 112. For this matrix the reduction of the number of iterations is not balancing the increase in flops. For instance, the number of iterations for D-Odir(8) is 695, and for D-Odir(12) it is 665, which represents a decrease of only 4% in the number of iterations. According to Theorem 1, it is very likely that in this case the preconditioned matrix does not have a cluster of small eigenvalues, hence the convergence of the method is not significantly improved when enlarging the Krylov subspace. However, we also notice that using the dynamic Orthodir variant (D-Odir) allows to reduce significantly the runtime when t is large: D-Odir is around 20% faster than Odir.

We also perform this study for Ela_20 with 112 MPI processes. First we remark that a breakdown occurs with Orthomin for all the values of t that we tested. This behavior of elasticity matrices had also been reported in [19]. Using BF-Omin effectively cures the breakdowns but does not allow the method to converge within the prescribed maximum number of iterations. Similarly, D-Odir does not converge when $t = 4$, but performs very well when t is larger. On the contrary, Odir is very stable and converges for all the values of t tested. As for Bump_2911 we observe that D-Odir is around 20% faster than Odir when $t = 24$. Overall, for this matrix, the best method is D-Odir with $t = 24$.

In order to better understand how the dynamic reduction of the search directions affects the convergence we plot the normalized residual and the block size (dash line) as a function of the iteration count for Flan_1565 (Fig. 4a) and Ela_20 (Fig. 4b). We notice that the convergence is not really affected by the reduction of the search directions because the number of iterations remains almost the same. However the block size is effectively reduced as soon as the method starts to converge. We note that the Ela_20 test case is very favorable: the block size is reduced even a bit before the convergence and the number of

	t	Odir	D-Odir	Omin	BF-Omin
Flan_1565	1	56.9	62.8	56.7	56.7
	4	36.3	36.4	35.5	35.9
	8	30.0	29.6	29.0	29.1
	12	30.2	29.1	29.8	29.4
	16	31.3	29.3	30.2	30.7
	20	33.1	30.7	32.0	32.7
	24	37.9	33.7	36.2	36.9
	28	39.2	34.9	37.6	38.5
Bump_2911	1	54.4	53.3	53.4	53.0
	2	64.9	62.7	64.5	65.5
	4	76.9	72.4	75.4	77.0
	8	93.6	85.4	91.5	91.5
	12	123.1	104.1	122.1	122.5
	16	151.2	123.6	147.1	148.6
	20	179.7	143.3	174.0	178.4
	24	198.3	158.3	195.5	199.3
	28	223.6	171.8	219.0	223.5
Ela_20	1	++	++	++	++
	4	97.6	++	-	++
	8	72.8	55.0	-	++
	12	56.8	51.5	-	++
	16	53.6	47.5	-	++
	20	56.3	47.2	-	++
	24	57.8	46.6	-	++
	28	59.9	47.5	-	++

Table 3: Runtime results (in seconds) for Flan_1565 ($P = 56$), Ela_20 ($P = 112$) and Bump_2911 ($P = 112$). The ++ means that the maximum number of iterations (5000) was reached and the - means that a breakdown occurred.



(a) (b)

Figure 4: Convergence of D-Odir (odir-1) compared to Odir (odir-0). The dash line represents the number of search directions for D-Odir. In parenthesis the difference of iteration count to reach convergence between D-Odir and Odir (+ means that D-Odir took more iterations to converge).

iterations is lower than when the search reductions are not reduced.

In conclusion, D-Odir is the best method over the different variants of ECG that we tested: it is a good compromise between the stability of Odir and the efficiency of the classical CG. Nevertheless, there exists matrices such as Bump_2911 for which the reduction of the number of iterations does not compensate the extra cost of ECG compared to the classical CG, even when using the dynamic reduction of the search directions. These results support the theoretical convergence study that was done in the previous section. ECG(t) is acting as if the t smallest eigenvalues of the matrix were deflated. Finally, we notice that values of t between 8 to 24 are good default parameters. Indeed, such values allow to effectively reduce the number of iterations while maintaining an affordable cost per iteration.

4.3.2 Strong scaling study

Following the parameter study, we perform a strong scaling study on Flan_1565 and Ela_30. As Bump_2911 does not seem particularly well suited for the method we do not perform the strong scaling study on this matrix.

For Flan_1565 we compare PETSc PCG and D-Odir with $t = 12$, the best choice over the parameters we tested. The resulting runtimes are presented in Table 4. When the number of MPI processes is relatively low ECG scales as well as PETSc, *i.e.*, almost linearly. As the number of iterations is significantly reduced with D-Odir(24), there is about 20% speed-up compared to PETSc at such scales. Nevertheless, we notice that for 2,016 MPI processes PETSc is significantly faster than ECG. This is likely because the number of iterations with PETSc is reduced with respect to 1,008 MPI processes. This behavior is not expected because it is known that block Jacobi preconditioners are not scalable (see [12] for instance). Indeed, we observe that the number of iterations is

	D-Odir(12)		PETSc CG		
# MPI	# iter	time (s)	# iter	time (s)	speed-up
252	332	12.0	1,709	14.8	1.2
504	405	6.1	2,430	8.4	1.4
1,008	519	4.1	3,179	4.9	1.2
2,016	637	3.6	2,687	2.6	0.7

Table 4: Strong scaling study for Flan_1565. The speed-up is the ratio between PETSc runtime and ECG runtime.

	D-Odir(24)		PETSc CG		
# MPI	# iter	time (s)	# iter	time (s)	speed-up
252	513	77.9	13,626	406.8	5.2
504	531	45.5	15,819	258.9	5.7
1,008	606	23.7	17,023	94.7	4.0
2,016	696	14.5	19,047	34.5	2.5

Table 5: Strong scaling study for Ela_30. The speed-up is the ratio between PETSc runtime and ECG runtime.

effectively increasing both for ECG and CG when the number of MPI processes increases.

Then we make the same comparison on Ela_30 test case for which we use D-Odir and $t = 24$, as discussed previously. The resulting runtimes are summarized in Table 5. We observe that both PETSc and ECG are scaling very well. Enlarging the Krylov subspaces allows us to reduce drastically the number of iterations: D-Odir(24) performs around 25 times less iterations than CG. As a consequence, D-Odir(24) is more than 5 times faster than PETSc PCG at small scale and around 2.5 times faster at large scale. We believe that this relatively poor scaling of D-Odir(24) compared to PETSc PCG at large scale is due to the implementation that is not as optimized as PETSc which has been developed for many years. For instance, the routine we use for computing the sparse matrix–set of vectors multiplication is certainly not as optimized as that of PETSc for computing the sparse matrix–vector multiplication. Also, we mentioned that we are currently performing 4 calls to `MPI_Allreduce` per iteration, but that could be reduced to 2 by fusing them. Furthermore, we could use Pipelining [16] or Communication-Avoiding based on s-step methods [23, 5] on top of ECG — that would require to take into account a possible loss of numerical stability of the method.

			D-Odir		PETSc CG		
# MPI	n	t	# iter	time (s)	# iter	time (s)	sp.-up
252	6.15×10^5	24	323	4.5	8,842	13.2	2.9
504	1.21×10^6	24	418	6.9	11,652	18.3	2.7
1008	2.38×10^6	24	538	9.8	14,487	24.6	2.5
2016	4.61×10^6	24	696	14.5	19,047	34.5	2.5
252	6.15×10^5	12	506	4.2	8,842	13.2	3.1
504	1.21×10^6	16	536	6.4	11,652	18.3	2.9
1008	2.38×10^6	20	538	9.7	14,847	24.6	2.5

Table 6: Weak scaling study. The dimension of the matrix is denoted n, and t denotes the enlarging factor. The speed-up (sp.-up) is the ratio between PETSc runtime and ECG runtime.

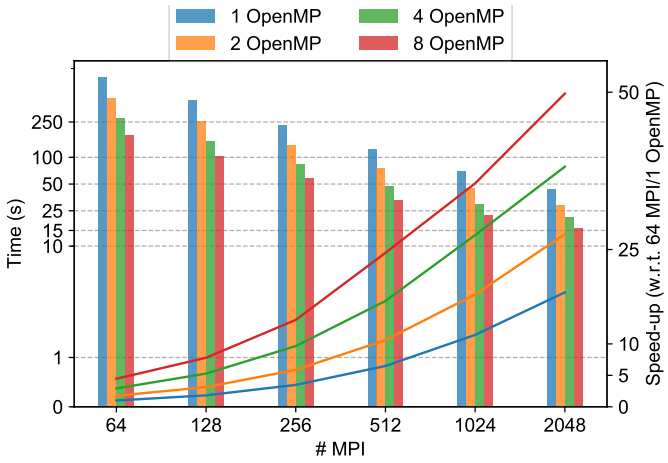
4.3.3 Dependence on the mesh size

Given the importance of the parameter t regarding the efficiency of the method, we perform a study of the convergence of the method with respect to the mesh size for the elasticity test case. More precisely, we consider the mesh used for generating the Ela_30 matrix, then we coarsen it by dividing the number of points in each dimension by $2^{1/3}$. Thus, we generate 3 additional elasticity matrices on which we perform a weak scaling experiment. Our major focus is not the weak scaling of ECG, but rather the comparison between PETSc's CG and D-Odir in terms of runtime.

The results are summarized in Table 6. First, we fix $t = 24$ and we perform a weak scaling study. We observe that D-Odir(24) is always between 2.5 to 2.9 times faster than PETSc PCG, but the gap tends to slightly decrease when the number of MPI processes increases. As ECG(t) is acting as if the t smallest eigenvalues of the matrix were deflated, it seems natural to use smaller values of t for the smaller matrices. Indeed, we perform another set of experiments where we vary t as well as the size of the problem. We observe that this slightly improves the results for the smaller matrices, we obtain for example an overall speed-up of 3.1 compared to 2.9 for the smallest matrix. As expected, this does not have a significant effect on the larger matrices. Another interesting observation is that even if the number of iterations is almost constant when increasing both t and the number of MPI processes, it does not improve the scaling because the cost of one iteration also increases. Overall, D-Odir is still at least 2.5 times faster than PETSc CG.

4.3.4 Impact of threads on performance

One motivation for enlarging the Krylov subspaces is to increase the arithmetic intensity of the resulting methods. This is particularly interesting to take advantage of the so-called manycore architecture as Nvidia GPUs, Intel Xeon Phi,



(a) The bars represent the runtime (left) and the lines represent the corresponding speed-up with respect to 64 MPI with 1 thread each (right).

	Omin(1)		Odir(24)	
# omp	time (s)	speed-up	time (s)	speed-up
1	89	-	44	-
2	74	1.2	29	1.5
4	80	1.1	21	2.1
8	79	1.1	16	2.8

(b) Comparison between Omin(1) and Odir(24) with 2048 MPI processes. We indicate the speed-up when increasing the number of threads for each method.

Figure 5: Strong scaling study for Ela_30 matrix on Cori (# omp stands for the number of threads assigned to each MPI processes).

or Sunway SW26010 used in the Sunway TaihuLight supercomputer. As the implementation relies on the MKL library which is multi-threaded [39], it is straightforward to assess its efficiency on the Xeon Phi processors.

In order to do so, we perform the following experiments on NERSC's supercomputer Cori. It consists in two partitions, one with Intel Haswell processors and another one with the last generation of Intel Xeon Phi processors: Knights Landing (KNL). More precisely, the second partition consists in 9,688 single-socket Intel Xeon Phi 7250 (KNL) processors with 68 cores each. For a detailed description of the machine, we refer to the online documentation[2]. We compile the code (and its dependencies) using the default compilers and libraries installed on the machine: `icc` version 18.0.1, `cray-mpich` version 7.6.2, MKL version 2018.1.163 and METIS version 5.1.0. We consider Ela_30 test case and we study the impact of threads on the strong scaling of Odir(24). We do not use the dynamic reduction of the search directions in order to keep the cost of one iteration constant during the solve to better understand the effect of threading. We both increase the number of MPI processes from 64 to 2048 and the number of threads from 1 to 8 – this means at most $2,048 \times 8 = 16,384$ threads, each one being bound to one physical core.

The results obtained are summarized in Fig. 5a. First of all, we notice that there is a trade-off between using threads or MPI processes because the number of MPI processes dictates the preconditioner. Indeed, there are as many blocks in the block Jacobi preconditioner as the number of MPI processes, thus increasing the number of MPI processes deteriorates the quality of the preconditioner. For instance, using 512 MPI processes takes 123s, and using 64 MPI processes with 8 threads each takes 179s: it is an increase of 50% compared to 512 MPI processes. Nevertheless, we observe that using more than 2 threads, and up to 8, has always a significant effect on the speed-up, even when the number of MPI processes is high. For instance, as shown in Table 5b, increasing the number of threads from 1 to 8 with a fixed number of $2,048$ MPI processes leads to an decrease in runtime of nearly 3. Of course, we are not close to full efficiency when using multiple threads, but we are still taking advantage of the BLAS 3 routines. This is illustrated by the Table 5b where we compare the speed-up obtained by using threads for Omin(1), which corresponds to the classical PCG, and Odir(24). We observe that using more than 2 threads is not effective at all with Omin(1) whereas it always significantly increases the speed-up with Odir(24).

Finally, we are able to obtain an overall speed-up of 50 when using $16,384$ cores with respect to 64 cores. Compared to an ideal speed-up of 256, it may seem that this result is not very good (around 20% of efficiency), however it is well-known that Krylov methods may face efficiency issues at very large scale[3] — in practice, such difficulties are overcome by using preconditioning strategies well adapted to the underlying problem. Furthermore, it is important to notice that the matrices tested are relatively small, but they allow us to simulate

[2] `http://www.nersc.gov/users/computational-systems/cori/configuration/`

[3] This is well illustrated by HPCG benchmark: `http://www.hpcg-benchmark.org/custom/index.html?lid=155&slid=293`

extreme scale situation: with $16,384$ cores the average number of unknowns per core is around 280. We have shown that in such cases, using enlarged Krylov subspaces allows to increase arithmetic intensity while decreasing the communication by drastically decreasing the overall iterations. Thus it takes advantage of the current trend in hardware architecture for reaching exascale.

5 Conclusion

In this paper, we have studied the Enlarged Conjugate Gradient method. It relies on so-called enlarged Krylov subspaces which can be seen as particular cases of block Krylov subspaces. The parallel efficiency of the approach has been assessed, and we have shown that this method is scalable up to $16,384$ cores and it is up to 5.7 times faster than PETSc's implementation of PCG.

First, we have thoroughly studied the method from a theoretical point of view. In particular, we have described the link between the two variants of the method (Orthodir and Orthomin), and thus explained their differences in terms of robustness. We also have studied its convergence rate and we have showed that the Enlarged Conjugate Gradient method is acting as if the smallest eigenvalues of the matrix were somehow deflated. Moreover, we have presented dynamic versions of Orthodir and Orthomin where the number of search directions is adaptively reduced during the iterations. Starting from the theory, we have described the link between the two variants of the method, and thus explained their differences in terms of robustness. Then, we studied its convergence rate and we have showed that ECG is acting as if the smallest eigenvalues of the matrix were somehow deflated. Then, we have described the parallel design of the method, including the two variants as well as their dynamic versions. Numerical experiments show that enlarging the Krylov subspaces allow to reduce significantly the number of iterations with respect to the classical PCG method. Furthermore, the reduction of the search directions allows to reduce the cost of the extra arithmetic operations induced by the method. Overall, the proposed solver is up to 5.7 times faster than PETSc PCG for an elasticity matrix. Also, as our implementation is based on BLAS 3 kernels only, it offers a good scaling when increasing the number of threads per MPI process. Thus, it is scaling up to $16,384$ threads, each one binded to a physical core, and it seems well adapted for so-called manycore architectures.

Throughout this work, the only assumption we make is that the matrix is symmetric positive definite. Hence the resulting methods are very generic and completely algebraic. For instance, it is straightforward to use D-Odir for solving linear systems with several right-hand sides. Similarly, ECG can be used with any preconditioner that could be used with CG. Thus, it can be used as a black-box solver that can be integrated very easily in any existing code. As it increases the arithmetic intensity and reduces the communication, it is well suited for modern and future architectures that exhibit massive parallelism. According to the theoretical study and the numerical experiments, it is particularly well adapted for matrices with a small number of low eigenvalues.

Acknowledgments

The computational resources were provided by the High Performance Computing Center North (HPC2N) at Umeå (Sweden), and the National Energy Research Scientific Computing Center (NERSC) at Berkeley (US), which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] E. Agullo, L. Giraud, and Y.-F. Jing. Block gmres method with inexact breakdowns and deflated restarting. *SIAM Journal on Matrix Analysis and Applications*, 35(4):1625–1651, 2014.

[2] S. F. Ashby, T. A. Manteuffel, and P. E. Saylor. A taxonomy for conjugate gradient methods. *SIAM Journal on Numerical Analysis*, 27(6):1542–1568, 1990.

[3] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, D. A. May, L. Curfman McInnes, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.8, Argonne National Laboratory, 2017.

[4] H. Calandra, S. Gratton, R. Lago, X. Vasseurm, and L. M. Carvalho. A modified block flexible GMRES method with deflation at each iteration for the solution of non-hermitian linear systems with multiple right-hand sides. *SIAM J. Sci. Comput.*, 35:345–367, 2013.

[5] E. Carson, N. Knight, and J. Demmel. Avoiding communication in nonsymmetric lanczos-based krylov subspace methods. *SIAM Journal on Scientific Computing*, 35(5):S42–S61, 2013.

[6] J. Chen. A deflated version of the block conjugate gradient algorithm with an application to gaussian process maximum likelihood estimation. Preprint P1927-0811, 2011.

[7] A. T. Chronopoulos. s-step iterative methods for (non)symmetric (in)definite linear systems. *SIAM Journal on Numerical Analysis*, 28(6):1776–1789, 1991.

[8] T. A. Davis and Y. Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, December 2011.

[9] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Communication-optimal parallel and sequential qr and lu factorizations. *SIAM Journal on Scientific Computing*, 34(1):A206–A239, 2012.

[10] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick. Minimizing communication in sparse matrix solvers. In *Proceedings of the ACM/IEEE Supercomputing SC9 Conference*, 2009.

[11] J. W. Demmel, L. Grigori, M. Gu, and H. Xiang. Communication avoiding rank revealing qr factorization with column pivoting. *SIAM Journal on Matrix Analysis and Applications*, 36(1):55–89, 2015.

[12] V. Dolean, P. Jolivet, and F. Nataf. *An introduction to domain decomposition methods*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2015. Algorithms, theory, and parallel implementation.

[13] Z. Dostál. Conjugate gradient method with preconditioning by projector. *International Journal of Computer Mathematics*, 23(3-4):315–323, 1988.

[14] A.A. Dubrulle. Retooling the method of block conjugate gradients. *ETNA. Electronic Transactions on Numerical Analysis [electronic only]*, 12:216–233, 2001.

[15] A. el Guennouni, K. Jbilou, and H. Sadok. A block version of bicgstab for linear systems with multiple right-hand sides. *ETNA. Electronic Transactions on Numerical Analysis [electronic only]*, 16:129–142, 2003.

[16] P. Ghysels and W. Vanroose. Hiding global synchronization latency in the preconditioned conjugate gradient algorithm. *Parallel Computing*, 40(7):224 – 238, 2014. 7th Workshop on Parallel Matrix Algorithms and Applications.

[17] L. Grigori, S. Moufawad, and F. Nataf. Enlarged Krylov Subspace Conjugate Gradient Methods for Reducing Communication. *SIAM Journal on Scientific Computing*, 37(2):744–773, 2016. Also as INRIA TR 8266.

[18] L. Grigori, F. Nataf, and Soleiman Y. Robust algebraic Schur complement preconditioners based on low rank corrections. Research Report RR-8557, Jul 2014.

[19] L. Grigori and O. Tissot. Reducing the communication and computational costs of enlarged krylov subspaces conjugate gradient. Research Report RR-9023, Feb 2017.

[20] M. H. Gutknecht. Block Krylov space methods for linear systems with multiple right-hand sides: an introduction. *in: Modern Mathematical Models, Methods and Algorithms for Real World Systems (A.H. Siddiqi, I.S. Duff, and O. Christensen, eds.)*, pages 420–447, 2007.

[21] F. Hecht. New development in freefem++. *J. Numer. Math.*, 20(3-4):251–265, 2012.

[22] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards.*, 49:409–436, 1952.

[23] M. Hoemmen. *Communication-avoiding Krylov subspace methods*. PhD thesis, University of California, Berkeley, 2010.

[24] A.Kalhan J. Dongarra, V.Eijkhout. Reverse communication interface for linear algebra templates for iterative methods. Technical report, May 1995.

[25] H. Ji and Y. Li. A breakdown-free block conjugate gradient method. *BIT Numerical Mathematics*, 57(2):379–403, Jun 2017.

[26] P. Jolivet, F. Hecht, F. Nataf, and C. Prudhomme. Scalable domain decomposition preconditioners for heterogeneous elliptic problems. In *Proceedings of the 2013 International Conference on High Performance Computing, Networking, Storage and Analysis*, SC13, pages 80:1–80:11. ACM, 2013.

[27] P. Jolivet and P. Tournier. Block iterative methods and recycling for improved scalability of linear solvers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16, pages 17:1–17:14, Piscataway, NJ, USA, 2016. IEEE Press.

[28] Pierre Jolivet. *Domain decomposition methods. Application to high-performance computing*. Theses, Université de Grenoble, October 2014.

[29] G. Karypis and V. Kumar. Metis – unstructured graph partitioning and sparse matrix ordering system, version 2.0. Jan 1995.

[30] M. Kreutzer, J. Thies, M. Röhrig-Zöllner, A. Pieper, F. Shahzad, M. Galgon, A. Basermann, H. Fehske, G. Hager, and G. Wellein. GHOST: Building blocks for high performance sparse linear algebra on heterogeneous systems. *International Journal of Parallel Programming*, 45(5):1046–1072, Oct 2017.

[31] J. Langou. *Iterative methods for solving linear systems with multiple right-hand sides*. PhD thesis, CERFACS, 2003.

[32] D. P. O'Leary. The block conjugate gradient algorithm and related methods. *Linear Algebra and Its Applications*, 29:293–322, 1980.

[33] M. Robbé and M. Sadkane. Exact and inexact breakdowns in the block GMRES method. *Linear Algebra Appl.*, 419:265–285, 2006.

[34] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[35] N. Spillane. Adaptive Multi Preconditioned Conjugate Gradient: Algorithm, Theory and an Application to Domain Decomposition. preprint, 2015.

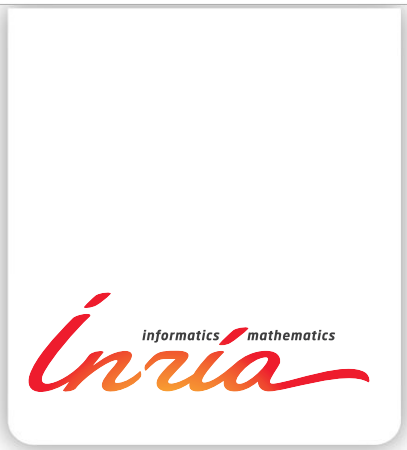
[36] N. Spillane. An adaptive multipreconditioned conjugate gradient algorithm. 38:A1896–A1918, 01 2016.

[37] N. Spillane, V. Dolean, P. Hauret, F. Nataf, C. Pechstein, and R. Scheichl. Abstract robust coarse spaces for systems of pdes via generalized eigenproblems in the overlaps. *Numerische Mathematik*, 126(4):741–770, Apr 2014.

[38] R. Thakur, R. Rabenseifner, and W. Gropp. Optimization of collective communication operations in mpich. *The International Journal of High Performance Computing Applications*, 19(1):49–66, 2005.

[39] Endong Wang, Qing Zhang, Bo Shen, Guangyong Zhang, Xiaowei Lu, Qing Wu, and Yajuan Wang. Intel math kernel library. In *High-Performance Computing on the Intel® Xeon Phi*, pages 167–188. Springer, 2014.

[40] I. Yamazaki, S. Tomov, and J. Dongarra. Mixed-precision cholesky qr factorization and its case studies on multicore cpu with multiple gpus. *SIAM Journal on Scientific Computing*, 37(3):C307–C330, 2015.



RESEARCH CENTRE
PARIS – ROCQUENCOURT

Domaine de Voluceau, - Rocquencourt
B.P. 105 - 78153 Le Chesnay Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399